Приключения Понерки Ксении или 25 лет Спустя

Yuriy Lesyuk



Содержание

aopk25

Приключения Пионерки Ксении

1991

Часть I. Введение. Технология взлома

Если Вы хотите научиться плавать, - плавайте, если вы хотите научиться говорить по-английски, - разговаривайте. Народная мудрость. Я захотел научиться писать компьютерные игрушки коммерческого качества. И мне ничего другого не оставалось, как сделать это.

Конечно, фирмы-изготовители игрушек не очень-то рекламируют свои исходные тексты. Ну что же, для такого случая и существует Sourser. Безусловно, комментарий, который выдает этот дизассемблер, можно считать комментарием лишь с большой натяжкой. Ну что ж, для настоящего программиста ассемблерный код - самоочевиден.

Наверное самым сложным из всего, что случилось за те несколько недель, в которые проходила работа, было выбрать саму игрушку. С одной стороны, не хотелось огорчать фирму, секреты которой я невольно приоткрою. С другой: уж если ломать, то такое, чтобы самому было и приятно и полезно.

Выбор пал на "Капитана Комика" американца Майкла Денио (Captain Comic by Michael Denio).

По многим причинам. Это одна из самых симпатичных мне аркадных игрушек (arcade game). У нее классические характеристики:

```
- экран 320 x 200, 16 цветов;
- сюжетное построение;
- стандартный набор музыкальных возможностей: фоновое
  сопровождение, спец. эффекты,
```

и, что немаловажно, хорошо развиты графические возможности (спрайтовая графика, мультипликаты).

Что окончательно склонило выбор в пользу данного представителя капиталистического рынка программного обеспечения: у нее, что редко встречается движущееся панорамирование игрового пространства (согласитесь, мало того, что этот эффект просто смотрится, даже на глаз можно сказать, что сделать это не так-то просто.).

Конечно, это уже старая игрушка - еще 1989-го года. Но если внимательно проанализировать более свежих представителей этого жанра, то можно заметить, что по сути дела они становятся более изощренными (256 цветов, реалистическая графика и динамика поведения персонажей, совершенее звуковое сопровождение (real sound)). Но сам набор возможностей большого количества компьютерных игрушек функционально покрывается имеющимся в "Капитане Комике".

Самая важная причина, по которой мне хотелось разобраться в компьютерных играх: удручающий дилетантизм в методической литературе, посвященной им: общие рассуждения, примитивные примеры...

Результатом многочасовой (и даже - многодневной!) работы и является следующий цикл статей, который можно было бы подзаглавить "Техника реализации функциональных элементов компьютерных игрушек. Продвинутый этап (advanced period).

В данной статье мы кратко познакомимся с основными особенностями процесса взлома компьютерных игрушек.

Немного остановимся на конкретных трудностях и путях их преодоления.

Перечислим функциональные подсистемы "Капитана Комика" и пообещаем более полно рассказать об этих подсистемах в следующих статьях цикла.

Первое, с чего нужно начинать взламывать компьютерную игрушку - это получение двоичной копии опаративной памяти, содержащей в себе работающую программу, так называемый слайд выполнения (execution slide).

Аргументов здесь несколько.

Один из основных, - упаковка программ DOS-овской утилитой EXEPACK. Что это значит для программиста: что он имеет возможность уменьшить количество места, требуемого для хранения своей программы, кроме того, уменьшается время загрузки программы с диска и все это за счет сжатия областей, заполненных последовательностями одинаковых байтов в специальный формат.

(Во время запуска такого файла на выполнение, сначала отрабатывается фрагмент, раскручивающий закодированный формат, и уже затем управление передается самой программе.)

Для взломщика же это означает, что почти весь полученный после дизассемблирования загрузочного модуля текст игрушки оказывается непригодным для анализа.

Часть данных интерпретируется, как программный код. Часть же адресов и смещений полностью теряет свой смысл: информация по этим адресам не соответствует действительности.

Вот в этот момент и можно по достоинству оценить преимущество формирования ссылки в командах перехода 86 ассемблера относительно точки перехода: благодаря этому большие куски программы оказываются корректными!

Поскольку в нашем дизассемблерном листинге и так будет много темных мест, не стоит усугублять себе задачу лишней работой (то есть, пытаться дизассемблировать текст загрузочного модуля).

Лучше подумать над тем, как быстро сделать слайд программы.

Трассировать упакованную игрушку - бесполезно: можно часами созерцать как EXEPACK-овский раскрутчик повторяет один и тот же цикл.

Вычислить же начало истинной программы не всегда возможно. Кроме того, ставить точки прерывания по некоторым причинам (далее - более подробно) часто невозможно.

Один из вариантов: когда я проанализировал начало Комика, то по контексту (встретился код операции открытия файла) обнаружил, что Комик довольно таки просто обрабатывает необходимые ему вспомогательные файлы. Если операция открытия файла возвращает ошибку, Комик восстанавливает адреса прерываний и возвращается в систему.

Дальше все просто: уничтожив файл, содержащий экран заставки, запускаю (по Go) в отладчике Комика. Я действительно оказываюсь на нужном адресе. Остальное - дело техники и команды записи файла (W) отладчика, и готов файл - слайд памяти.

```
Для попадания внутрь работающей программы с целью получить слайд
памяти существует один довольно таки интересный способ: если мы
запускаем отладчик AFD в резидентный режим, а после запуска на
выполнение отлаживаемой (читай - взламываемой) программы
нажимаем Ctrl-ESC (код для выхова AFD), то последний честно
перехватывает управление и, ломая графику и адреса векторов
прерываний, дает нам возможность записать память не диск.
```

Другой аргумент получения слайда памяти выполнения программы, он же - ьнекоторая причина невозможности трассировки программыь, упомянутая выше,

- INT 3. Оказывается, по тексту программы раскидан INT 3! А через INT 3, оказывается, Комик работает со своим музыкальным монитором.

Для тех счастливцев, кто не знает, что такое INT 3, объясняю: программные прерывания как таковые (а INT 3 является программным прерыванием), грубо говоря, эквивалент подпрограмм. Все прерывания - двухбайтные. Но одно сделано обнобайтным. Это INT 3. Предназначено оно для целей организации отладчиками процесса работы с точками остановов: с адреса точки останова отладчик списывает байт, и на место этого байта записывается код INT 3. По самому же INT-у ставится переход на отладчик. Далее по команде Go пользовательской программе передается управление, и последняя работает до тех пор пока не достигает команды INT 3, которая и возвращает управление отладчику.

Очень удобно, особенно если учесть, что один байт это все таки не 2. Вот это удобство и используют все отладчики.

При запуске Комика, последний изменяет содержимое адреса прерывания INT 3, что в отладчике и приводит к зависанию машины.

```
Конечно, с точки зрения Майкла Денио, это быть может и кажется
остроумной защитой от взламывания, я же поняв, что услугами
отладчиков мне уже воспользоваться не придется, хорошей такую
идею счесть, безусловно, не мог.
```

Еще одно рассуждение о пользе слайдов памяти: многие предварительные установки, инициализации исходных переменных, фиксации состояний игры, будучи зафиксированными на слайдах памяти, оказывают дополнительную помощь в процессе осмысливания, то есть, восстановления смысла работы программы.

После того, как мы получаем слайд выполняемой памяти программы, к нашим услугам прекрасная программа - Sourser (дай бог здоровья ее создателям: скольким программистам они сьекономили и его и времени!).

Все, что происходит между получением листинга после Sourser-а и перед окончательным редактированием взламываемой вами программы - покрыто мраком. Это работа творческая. Одно могу отметить точно: тот, кто сказал, что проще написать свою программу, чем разобраться в чужой, явно приврал: на самом деле написать свою программу в такой ситуации во много раз (!!!) проще, а не просто ьпрощеь.

```
Общими словами здесь, конечно, не обойтись, поэтому многие из
встретившихся ситуаций психологической интерпретации кода будут
проанализированы и описаны в следующих статьях, на конкретных
примерах.
```

Основное правило разбора в непрокомментированных текстах: после того, как получен листинг дизассемблера, компьютер до-о-лго не требуется: достаточно только стол и ручка.

Еще раз вернемся к процессу трассировки. Да, безусловно, во многих случаях трассировка как таковая, во много раз повышает эффективность анализа программы. Видеть, что можно проверить свою гипотезу одной-единственной точкой останова и не иметь такой возможности, ну оч-чень обидно!

Возможность достойно выйти из такой ситуации дает прекрасный отладчик PERISCOPE. Чем же этот отладчик так прекрасен?

Он позволяет реализовывать пользовательские прерывания (user interrupt). В частности, одно из написанных мною прерываний организует программую точку останова через нужный мне двухбайтный INT. (О технике работы с

точками останова было рассказано выше.)

Другое свое (пользовательское) прерывание, которым я часто пользуюсь позволяет мне отлаживать программы, используемые графические режими. В чем дело? Если Вы отлаживали хоть какую-то графическую программу, Вы не могли не заметить: вернулись в отладчик, вернулись в программу, а на экране - грязные пятна вместо половины экрана...

Диагноз прост: одладчик пользуется той же ОЗУ-ой (оперативной памятью), что и наша программа.

Курс лечения не намного сложнее: возвращаем прерывание отладчику после того, как скопируем в буфер фрагмент ОЗУ экрана пользовательской программы, который портится, после отладчика - восстанавливаем это ОЗУ.

И последнее, но не маловажное, о чем бы хотелось сказать. Об инструменте, который должен быть в арсенале взломщика программ, то есть, настоящего программиста. Значение его (инструмента) трудно даже и переоценить: он как фомка, как отмычка (для другого взломщика.

Инструмент этот: таблица, в порядке возрастания кодов то второй колонке которой соответствующая им ассемблерная команда.

```
Для 86-го процессора - попытайтесь вспомнить - такой таблицы
нет, а ведь после дизассемблера - попробуйте возразить - нередко
приходится самому поработать дизассемблером.
```

Изготовить такую штуку, конечно, просто. в отладчике набираем строки:

```
00 90 90 90 90 90
01 90 90 90 90 90
02 90 90 90 90 90
...
ff 90 90 90 90 90
```

Эти нопы, естественно, нужны, потому что первый байт в команде - не обязательно единственный. А дальше - дизассемблируем (по U) набранный текст, и наша фомка - готова!

Музыкальный проигрыватель. Статья посвящается технической стороне реализации музыкального сопровождения в игрушке "Капитан Комик".

Кратко рассмотрен процесс звукоизвлечения при помощи компьютера. Рассказано о нюансах, связанных с необходимостью получать звук в фоновом режиме. Впрочем, эту информацию мы можем почерпнуть и из других литературных источников (список которых, конечно же, приводится).

Более интересным является рассмотрение процесса взаимодействие музыкальной части с остальной программой игры: технология запуска мелодий, включение/отключение звука, реализация механизма приоритера мелодий, получение программой информации о состоянии мелодии.

Материал полностью проиллюстрирован листингами, блок-схемами. Кроме того, приводится реализация функций, позволяющий организовать музыкальное сопровождение в любой пользовательской системе.

Как пример, дается драйвер музыкального проигрывателя для систем, написанных на языке dBASE и ему подобных.

Драйвер ввода. Клавиатура. Немаловажный момент: процесс ввода информации в машину приобретает особое значение в игрушках.

Статья рассматривает вопросы опроса клавиатуры, проблемы фиксации нажатия/отпускания различных клавиш. Анализируются проблемы ввода информации с устройств типа джойстик/мышь. Приведены реализации конечных автоматов, позволяющих организовать функциональное соответствие устройств клавиатуре.

Конечно, все проиллюстрировано фрагментами программ, рисунками.

Драйверы графики: заставки, спрайты, панорамирование. Пожалуй, самый ответственный момент любой компьютерной игры - графика.

И каких только технических ухищрений не используют при работе с экраном. Ведь счет времени идет на миллисекунды! Некоторые из этих ухищрений рассматриваются в данной статье. Анализируемые примеры и программы реализуют графические драйвера для адаптеров типа EGA/VGA.

Описаны процессы реализации спрайтовой графики. Приведены примеры анализа функциональной части игры, и коды программ, в которые выливается этот анализ (здесь, в частности, и рассмотрено панорамирование).

Далее рассматриваются спецэффекты графики: мультипликация (явление Комика не пранету, дематериализация Комика), мигание (призы в уголке экрана), взблескивание (в первом и втором экранах заставки).

Безусловно, техническая информация о графических адаптерах, операционной системе только повышает полезность статьи.

Технология программирования компьютерной игры: идея, сюжет, реализация. Пожалуй, самое интересное, о чем реже всего предпочитают говорить книги и учебники, это организация работы при проектировании и программировании самой компьютерной игры.

Приводится примерный план процесса реализации игрушки с учетом всех технических этапов; для каждого этапа приводится инструментарий, позволяющий облегчать построение элементов игры (графические/музыкальные редакторы, мультипликаторы, встроенные средства отладки игр, специализированные языки программирования игрушек и т.п.).

Здесь происходит знакомство с функциональной схемой работы "Капитана Комика", основными ее элементами. Приводится специфика отладки программ, работающих в режиме реального времени и реализующих динамику и элементы многопроцессности.

Воссозданная последовательность поэтапной работы над программой "Капитан Комик", а также расклад работ по времени завершают изложение.

Музыкальный Проигрыватель

Как я взламывал Капитана Комика

Часть II. Музыкальный проигрыватель

Механизм, реализующий музыкальное сопровождение "Капитана Комика" работает следующим образом.

Резидентная программа music_player, висящая на 8-ом прерывании (прерывании таймера, которое обрабатывается каждые 55 миллисекунд) 18.2 раза в секунду анализирует содержимое специальной области данных. И в зависимости от результатов анализа производит некоторые действия.

Если в данный момент нет активной мелодии, просто передает управление оригинальной программе обработки 8-го прерывания.

В случае, когда мелодия активна, анализируется длительность текущей ноты. (Конечно, ноты, заданной в виде частоты). Если длительность ноты отработана, выбирает следующую ноту.

Если выбирается величина 0000h, выключает мелодию, сбрасывая значение соответствующих переменных.

После того, как таймер 8253 запрограммирован на требуемую новой нотой частоту, анализируется признак включенности звука. И, если звук включен, то устанавливается бит 8255 схемы, разрешающий динамик.

Установка значений переменных области данных производится программкой music_monitor. Передаваемая через регистры информация интерпретируется следующим образом:

```
ax - код операции;
cx - приоритет мелодии;
bx - адрес мелодии.
```

Что касается адреса мелодии. Поскольку в момент вызова не изменяется состояние регистра ds, то мы можем фиксировать ds как сегмент мелодии, следовательно, сама мелодия мелодия может располагаться в любом месте оперативной памяти.

Содержимое региста cx позволяет музыкальному монитору реализовать механизм приоритета мелодий: монитор сравнивает приоритет текущей мелодии с приоритетом загружаемой мелодии. И если последний выше, то происходит запуск загружаемой мелодии, если нет, - отрабатывается старая мелодия.

```
Обратим внимание: если мы в самой игрушке проходим границу
панорамы, раздается характерная мелодия. Попробуем в момент
игры этой мелодии нажать клавишу  стрельбы (Insert) и убедиться,
что пока мелодия не доиграет, звука выстрелов не слышно, хотя
сам выстрел на экране происходит.
```

В регисте ax передается код операции, по которому монитор устанавливает значение некоторых переменных, считываемых резидентом. Поскольку операции монитора и конкретные переменные тесно взаимосвязаны, опишем область данных, обслуживающих музыкальный монитор, привязывая их к операциям.

```
old_int_8 - переменная типа dd (двойное слово); содержит старое
    (оригинальное) значение адреса перехода на программу
    обработки 8-го прерывания.

melody_status - переменная (байт) содержит статус мелодии;
    операция 1 - выбрать новую мелодию - устанавливает
    значение данной переменной в On (числовой
    эквивалент - 1);
    операция 3 - выключить мелодию - сбрасывает значение
    этой переменной в Off (числовой эквивалент - 0);
    также в значение Off данная переменная устанавливается
    программой music_player когда мелодия завершается
    естественным путем;
    операция 4 - возвратить статус мелодии - возвращает в
    регистре ah значение этой переменной, что позволяет
```

```
    пользовательской программе, например, дождатьсся конца
    мелодии.

sound_status - переменная (байт) содержит статус звука;
    операция 0 - включить звук устанавливает переменную в
    значение On;
    операция 2 - выключить звук устанавливает переменную в
    значение Off;
    функциональные клавиши f1, f2 (запретить/разрешить звук)
    приводят к воздествию именно на эту клавишу.

melody_priorty - приоритет мелодии (байт);
    переменная содержит значение приоритета мелодии; чем
    больше число, тем больший приоритет и, таким образом,
    для мелодий с приоритетами 1 и 4 будет выполняться
    вторая.

melody_seg,
ptr_crnt_note - сегмент мелодии, указатель на текущую ноту -
    пара переменных (слово, слово) содержит 4-рех байтный
    указатель на текущую ноту; операция 1 - установить новую
    мелодию - устанавливает данные значения.

len_crnt_note - длительность текущей ноты - переменная (слово)
    содержит остаток длительности текущей ноты, который
    необходимо отыграть с данной частотой.
```

Теперь мы может проанализировать алгоритм, по которому функционирует music_player

```
                    |
        сохранить регистры
                    |
        загрузить сегмент мелодии
                     melody_seg
                    |               нет
                melody_status == On?      ------------------------
                    |
                декрементировать
```

```
                 len_crnt_note
           нет           |
   +____________ len_crnt_note == 0?
   |                     |
   |             увеличить указатель следующей
   |             ноты; выбрать след.ноту
   |                     |                  да
   |             следующая нота == 0000h?   ________________+
   |                     |                                  |
   |             выключить динамик                          |
   |                     |                                  |
   |             запрограммировать таймер                   |
   |             на генерацию частоты                       |
   |             [ptr_crnt_note]                            |
   |             сегмента melody_seg                        |
   |                     |                                  |
   |             выбрать длительность след.ноты             |
   |             и зафиксировать ее в                       |
   |             len_crnt_note                              |
   |                     |                                  |
   |     зддддд sound_status == On?                   выключить
   |     |               | нет                         мелодию
   |  включить           |                                  |
   |  дмнамик            +----------------------------------+
   |     |               +-----------------------------------------
   |     |               |
   |     |       выключить динамик
   |     +---------------+
   +---------------------+               восстановить регистры
                         |
                 передать управление по адресу
                 оригинального прерывания
                         |
                  ммммммммоммммммммм
```

Как же Капитан Комик работает с данным музыкальным проигрывателем?

В начале игрушки происходит установка вектора 8-го прерывания, запускающая программу music_player. После чего мы может вызывать программу music_monitor, допустим, следующим образом:

```
lea    bx, <смещение мелодии>
mov    ax, 1    ; операция - установить новую мелодию
mov    cx, 4    ; приоритет мелодии - в Комике
        ;              - самый высокий
call    music_monitor
```

Для того, чтобы воспользоваться статусом мелодии, фрагмент кода будет выглядеть следующим образом:

```
mov    ax, 4    ; код операции - получить статус мелодии
call    music_monitor
jnz    <адрес перехода в случае, если мелодия еще играет>
; команды, которые выполняются,
; когда мелодия уже отзвучала
...
```

Чтобы усложнить процесс взлома игрушки, Михаил Денио предпочитает вызывать music_monitor через INT 3, что не дает возможности протрассировать программу, либо поставить в ней точку останова (более подробно см. предыд. статью).

В этом случае, в процедуре инициализации векторов прерываний, третье прерывание устанавливается на программу music_monitor, а вызов программы выглядит так:

```
mov    ax, 2
int 3        ; для функции - выключить звук
```

Перед выходом из программы в операционную систему, безусловно, необходимо восстановить оригинальные значения прерываний.

~~~
```Screen # 0
( introduction                                             03/25/91 )
( +- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -+
  |                                                             |
  |    реализация игрушки Soko-ban предследует учебные          |
  |       цели и акцентирует внимание на критичных              |
  |         к процессу программирования местах                  |
  |                                                             |
  |                                                             |
  |                                                             |
  |                      программа продолжает славную           |
  |                      серию реализаций игрушки Soko-ban      |
  |                      на различных языках программиро-       |
  |                      вания от Insight corp.,                |
  |                      представляя Forth                      |
  |                                                             |
  + -- -- -- -- -- -- -- -- -- -- (c) Insight corp., 1991 --+ )

Screen # 1
( слова для ввода клавиши и позиционирования курсора 01/04/80 )


256  CONSTANT F_key ( системная константа -
                          смещение для скан-кодов )
: GETKEY ( --- n )  ( системный ввод )
  KEY
  DUP 0= IF DROP KEY F_key + THEN
;

: GOTO_XY ( x y --- )
  SWAP 2 * SWAP GOTOXY ;

: BELL ( --- ) 7 EMIT ; ( выдача звукового сигнала на консоль )

-->



Screen # 2
( определение констант                                     01/04/80 )
~~~

```
HEX
  ( коды клавиш управления )  1B   CONSTANT F_esc
                              20   CONSTANT F_restart
F_key 4B + CONSTANT F_Left    F_key 4D + CONSTANT F_Right
F_key 48 + CONSTANT F_Up      F_key 50 + CONSTANT F_Down
80   CONSTANT C_ctrl_bit
7F   CONSTANT C_bit_mask
           ( --- коды спрайтов игрушки --- )
0  DUP CONSTANT S_fre_plc     C_ctrl_bit + CONSTANT Sb_fre_plc
1  DUP CONSTANT S_man         C_ctrl_bit + CONSTANT Sb_man
9  DUP CONSTANT S_wall        C_ctrl_bit + CONSTANT Sb_wall
4  DUP CONSTANT S_box         C_ctrl_bit + CONSTANT Sb_box
DECIMAL
-->


Laboratory Microsystems PC/FORTH 2.0       21:24  10/11/90   rp.sc

Screen # 3
( игрушечные переменные                               03/23/91 )
  VARIABLE MAN_X   0 MAN_X !
  VARIABLE MAN_Y   0 MAN_Y !
  VARIABLE SCORE_ALL 0 SCORE_ALL ! ( максимальн.кол-во очков )
  VARIABLE SCORE_NEW 0 SCORE_NEW ! ( текущее кол-во очков    )
: SCORE_ALL++ ( --- ) SCORE_ALL @ 1 + SCORE_ALL ! ;
: SCORE_ALL-- ( --- ) SCORE_ALL @ 1 - SCORE_ALL ! ;
: SCORE_NEW++ ( --- ) SCORE_NEW @ 1 + SCORE_NEW ! ;
: SCORE_NEW-- ( --- ) SCORE_NEW @ 1 - SCORE_NEW ! ;
  VARIABLE KBRD_KEY  ( введенный с калвиатуры символ )
-->


Screen # 4
( определение типа ARRAY и определение поля игры     03/22/91 )
```

```
30 CONSTANT GF_col
20 CONSTANT GF_row
: ARRAY ( #col #row --- )
        SWAP CREATE OVER , * ALLOT
  DOES> ( #col #row --- A )
        DUP @ ROT * + + 2+ ;

GF_row GF_col ARRAY GF

: GF@ ( #col #row --- n ) GF C@ ;
: GF! ( n #col #row --- ) GF C! ;
: .GF ( \ cod-line ( #col #row --- )
  GF ASCII # WORD COUNT ROT SWAP CMOVE> ;

-->


Screen # 5
( инициализация игровой ситуации                    03/22/91 )
: GF_CLEAR ( -- ) GF_row 0 DO GF_col 0 DO 0 I J GF! LOOP LOOP ;
GF_CLEAR
0  1 .GF 00000009999000000000000000000000#
0  2 .GF 00000009000900000000000000000000#
0  3 .GF 00000009000900000000000000000000#
0  4 .GF 00000009400900000000000000000000#
0  5 .GF 00000999004999000000000000000000#
0  6 .GF 00000900400409000000000000000000#
0  7 .GF 00099909099909000009999990000000#
0  8 .GF 00090009099909999999003390000000#
0  9 .GF 00090400400000000000003390000000#
0 10 .GF 00099999099990919999003390000000#
0 11 .GF 00000009000000999009999990000000#
0 12 .GF 00000009999999000000000000000000#
-->


Laboratory Microsystems PC/FORTH 2.0       21:24  10/11/90   rp.sc

Screen # 6
( конвертер внешнего представления игры в внутреннее 03/23/91 )
```

```

: GF_CONVERT ( --- )
  GF_row 0 DO GF_col 0 DO I J GF@ CASE
         ASCII 0 OF S_fre_plc  ENDOF
         ASCII 9 OF S_wall     ENDOF
         ASCII 4 OF S_box      ENDOF
         ASCII 3 OF Sb_fre_plc SCORE_ALL++ ENDOF
         ASCII 7 OF Sb_box SCORE_NEW++ SCORE_ALL++ ENDOF
         ASCII 1 OF S_man    I MAN_X ! J MAN_Y ! ENDOF
         ASCII # OF S_fre_plc  ENDOF
         ( default ) S_fre_plc SWAP
  ENDCASE I J GF! LOOP LOOP ;
-->


Screen # 7
( вывод спрайта                                       03/23/91 )
: SPRITE_OUT ( n --- )
  CASE
    S_fre_plc     OF ."   " ENDOF
    Sb_fre_plc    OF ." . " ENDOF
    S_wall        OF 178 EMIT 178 EMIT ENDOF
    Sb_wall       OF 178 EMIT 178 EMIT ENDOF
    S_man         OF ." ><" ENDOF
    Sb_man        OF ." ><" ENDOF
    S_box         OF ." <>" ENDOF
    Sb_box        OF 17  EMIT 16  EMIT ENDOF
  ENDCASE ;
-->


Screen # 8
( системный вывод спрайта                             03/23/91 )
: SPRITE_SHOW ( x y t --- )
  >R 2DUP GF@ Sb_box = IF
```

```
    R> DUP >R S_box <> IF SCORE_NEW-- THEN THEN
  2DUP GF@ C_ctrl_bit AND 0 <> IF
    R> DUP >R S_box  = IF SCORE_NEW++ THEN THEN
  2DUP GF DUP C@ C_ctrl_bit AND R> OR DUP >R SWAP C!

  GOTO_XY R> SPRITE_OUT ;

: GF_SHOW ( --- )
  GF_row 0 DO GF_col 0 DO I J 2DUP GF@ SPRITE_SHOW LOOP LOOP ;

-->



Laboratory Microsystems PC/FORTH 2.0       21:24  10/11/90   rp.sc

Screen # 9
( слово MOVING - изюминка нашего дела                 03/23/91 )
: MOVING ( dlt_x dlt_y --- ) 2DUP
  >R MAN_X @ + R> MAN_Y @ + GF@ C_bit_mask AND S_fre_plc =
  IF   >R MAN_X @ + MAN_X ! R> MAN_Y @ + MAN_Y !
  ELSE 2DUP
       >R MAN_X @ + R> MAN_Y @ + GF@ C_bit_mask AND S_box =
       IF 2DUP >R 2 * MAN_X @ + R> 2 * MAN_Y @ +
          GF@ C_bit_mask AND S_fre_plc =
          IF   2DUP >R MAN_X @ + R> MAN_Y @ + S_fre_plc
               SPRITE_SHOW
               2DUP >R 2 * MAN_X @ + R> 2 * MAN_Y @ + S_box
               SPRITE_SHOW
               >R MAN_X @ + MAN_X ! R> MAN_Y @ + MAN_Y !
          ELSE 2DROP THEN
       ELSE 2DROP THEN
  THEN ;   -->



Screen # 10
( основной цикл игры                                  03/23/91 )
: GAME_ROUND ( --- )
  BEGIN
```

```
    MAN_X @ MAN_Y @ S_man     SPRITE_SHOW
    GETKEY KBRD_KEY !
    MAN_X @ MAN_Y @ S_fre_plc SPRITE_SHOW
    KBRD_KEY @ CASE
               F_Left  OF -1 0 MOVING ENDOF
               F_Right OF  1 0 MOVING ENDOF
               F_Up    OF 0 -1 MOVING ENDOF
               F_Down  OF 0  1 MOVING ENDOF
            ENDCASE
  SCORE_NEW @ SCORE_ALL @ =
     KBRD_KEY @ F_esc =
     KBRD_KEY @ F_restart = OR OR UNTIL ;
-->


Screen # 11
( запуск игры и раздача слоников                    03/23/91 )
: GAME_PUSHER ( --- )
GF_CONVERT
GF_SHOW       BELL BELL

GAME_ROUND

10 17 GOTO_XY BELL BELL BELL
SCORE_NEW @ SCORE_ALL @ =
IF    ." МАЛАДЕЦ "
ELSE  ." СЛАББАК "
THEN
10 19 GOTO_XY ;

GAME_PUSHER ( ...поехали! )


Laboratory Microsystems PC/FORTH 2.0       21:24  10/11/90   rp.sc
```

Одна игра на разных языках разные игры на одном языке

```
Бейсик, Паскаль, Си, Ассемблер, Форт, Фортран, диБейзIII, Солт,
Мульти-Эдит, Смолток. Пока достаточно. Можно ли назвать то
общее, что могло бы объединять эти языки. И если раньше этим
общим было только то, что все это - языки программирования, то
теперь к ним прибавилась еще одна черта: игрушка Soko-Ban,
которая далее будет реализована на всех этих языках.
```

Предисловие

Как-то раз, в молодости, восхищенный логически выверенными построениями лабиринтов игрушки Soko-Ban, я решил запрограммировать ее. Сделал это на Паскале. И, убедившись, что программа работает, отложил листинг куда-то в сторону. В самом деле, не продавать же его было американцам.

И вот некоторое время спустя произошла следующая история. Встретился с одним товарищем. Тот попросил рассказать о языке Паскаль. Я начал рассказывать, обяснил типы, операторы, процедуры, строение программы. А когда дело дошло до примера программы на языке вдруг вспомнил о когда-то написанном листинге с игрушкой Soko-ban. Быстренько введя друга в алгоритм функционирования игры, я тут же указывал на соответствующий фрагмент программы... А поскольку и игра и программа представляли собой завершенное построение, то мой рассказ о языке Паскаль не просто отложился в деталях в голове товарища, но сделал это [рассказ] в виде единого целого.

И, по-достоинству оценив методическую помощь, оказанную мне программой, в следующий раз другому товарищу я начал сразу же рассказывать о языке Паскаль на примере листинга.

Но это, оказалось было не все. И когда в одной ситуации, мне пришлось тому же товарищу, который уже знал Паскаль, объяснять язык Си, я снова прибег к помощи того же листинга игрушки Soko-Ban. Поскольку с алгоритмом игры он уже был знаком, как алгоритм был реализован на Паскале знал, мы бысто, указывая на общее и разное в обоих языках, наваяли программу на Си.

Более того, когда мне, как преподавателю необходимо было студентов, знающим Паскаль, обучать языку Ассемблера, мой старый листинг снова пригодился мне: я предложил им реализовать программу с языка Паскаль, на язык Ассемблера. А поскольку в данном случае в их задачу вошло еще и разобраться самим в алгоритме программы (фактически восстановить его из программы), то качество усвоения материала, я вас уверяю, увеличилось.

Вот таким образом обстояло дело исторически.

Что же имеем мы на сегодняшний день? И что представляет собой данная статья? Что вы, читатель, можете извлечь из нее полезного для себя?

Но по-порядку. Первый раздел статьи, не считая ее введения, представляет описание алгоритма игры Soko-ban, проанализированный с точки зрения сложности ее программной реализации.

Дальнейшие же разделы являют собой собственно программы - реализации игры на различных языках программирования. Вводные параграфы каждого "программного" раздела представляют язык, анализируют его характерные, особенности (конечно, кратко), и описывают конкретные нюансы реализации алгоритма игры на данном языке. Листинг же, следующий далее, подробно и откоментированно реализует игрушку.

Листинги разных игр похожи друг на друга в той же степени, в которой похожи одни на одного сами языки. Причем немногим более, чем поверхностный анализ программ уже показывает, что они представляют собой не переписанный алгоритм, реализованный на Паскале, к примеру, а в каждом языке, точнее в программе на соответствующем языке передан и сохранен дух самого языка, сохранены особенности технологии производства программ на данном языке и стилистика языка.

При описании программ и в их [программ] комментариях используется терминология, характерная для того или иного языка. Так, программа на Форте, состоит из экранов, определений слов. В реализации на Смолтоке приведен полностью протокол класса, инициализированы необходимые экземпляры и методы применительно к подмножеству реализации Смолтока Smalltalk/V. Причем предыдущий абзац вряд ли будет понят читателем, имеющем смутное представление о Форте или Смолтоке. Но тем, кто знает язык, и что важнее, тем кто тот или иной язык изучает, такой подход позволит проверять свое знание языка и понимание его.

Тот факт, что каждый язык предназначен для конкретных целей и решает эффективнее именно те задачи, для которых спроектирован, не только не оспоряется автором, но даже подтверждается каждой программой. (куда уж характернее пример программы на макро-языке редактора Мульти-Эдит, который, безусловно, хорош для написания макрокоманд обработки текста в среде редактора и заставил поломать голову при программировании вещи, совершенно не свойственной для редакторов текстов: написание программы с элементами интерактивного взаимодействия, являющейся к тому же компьютерной игрушкой). И тем не менее, такой подход оправдан. Ибо программисту, пользующемуся Мульти-Эдитом и решившему написать быстренько макро-команду, не имевшему ранее дело с макро-языками (допустим, Траком) это "быстренько" не только выльется в определенный (достаточно-таки длительный) период времени, но и отобьет всякую охоту заниматься этим в следующий раз. Данная же программа игрушки, потребовав на разбор не много, все-таки, времени, даст необходимый минимум знаний особенностей упомянутого макро-языка.

Таким образом, данное произведение следует воспринимать не только как курьез в области программирования, но и как справочник по языкам программирования в их сравнительном плане. (Более точно можно сказать: справочник по фразеологии языков программирования.)

Идея и реализация данной статьи чем-то перекликается с существовавшей в свое время (молодое поколение программистов вряд-ли помнит об этом, но было! было...) традицией писать на разных языках программирования

интроспективные программы, что позволяло судить о выразительной мощи языков. И следовательно языкам менее "выразительно мощным" не чувствовать себя ущербными.

Несколько слов о выборе языков программирования. В данном издании статьи будут приведены программы на тех языках, которые распространены сейчас в среде IBM PC-подобных машинах, то есть там, где не только не сложно (относительно) достать тот или иной компилятор того или другого языка, но и подбирались языки, все-таки распространенные в данной среде.

Алгоритм игры Soko-Ban

Сначала, почему именно данная игрушка удостоена такого пристального внимания? Во-первых, так сложилось исторически, как можно узнать из предисловия. Во-вторых, программа, используемая с данной целью (а именно

- с целью быть написанной на разных языках и приведенной в листингах полностью) должна удовлетворять ряду условий: быть не слишком большой, но и не слишком маленькой, быть не слишком сложной, но и не слишком простой. В-третьих, программа должна удовлетворять ряду общих требований: быть интересной, зрелищной и т.д.

Совокупность данных требований и позволила остановиться на игрушке Soko-Ban.

Принцип игры

Во всех версиях игра (и это было одним из требований технического задания) внешне выглядит следующим образом:

```
        ╠╠╠╠╠╠╠╠╠╠
        ╠╠      ╠╠
        ╠╠<>  <>╠╠
    ╠╠╠╠╠╠      ╠╠╠╠╠╠
    ╠╠    <>    <>  ╠╠
╠╠╠╠╠╠  ╠╠  ╠╠╠╠╠╠  ╠╠          ╠╠╠╠╠╠╠╠╠╠╠╠
╠╠      ╠╠  ╠╠╠╠╠╠  ╠╠╠╠╠╠╠╠╠╠╠╠╠╠    . . ╠╠
╠╠  <>    <>                          . . ╠╠
╠╠╠╠╠╠╠╠╠╠  ╠╠╠╠╠╠╠╠  ╠╠><╠╠╠╠╠╠╠╠    . . ╠╠
        ╠╠            ╠╠╠╠╠╠    ╠╠╠╠╠╠╠╠╠╠╠╠
        ╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠╠
```

Игра относится к классу логических головоломок и представляет собой лабиринт выложенный (во всех версиях данных реализаций) спрайтом, или изображением стенки ╠╠. Тут и там в лабиринте раскиданы или расположены изображения ящиков <>. А еще в лабиринте находятся изображения мест ящиков . . Главный герой игры >< (в меру - упитан, в меру - воспитан, в меру - симпатичен), часто в листингах называется двигателем, потому что двигает ящики, при помощи нажатия юзером клавиш Up, Left, Right и Down передвигается по лабиринту, натыкаясь на стенки и на ящики. Причем, если ящик один, а после него (в направлении по оси герой-ящик) находится свободное пространство, то при нажатии в данном направлении соответствующей клавиши, ящик сдвигается в том направлении. То есть, главный герой двигает ящики по лабиринту и пытается устанавливать их на места для ящиков. Если ящик становится на свое место, то он уже выглядит следующим образом шш (только повернутым на 45 градусов).

Вот цель-то игры и состоит в том, чтобы двигатель установил все ящики на соответствующие места. А ввиду ограничений, имманентных игре, (именно: если ящик подтолкнули к стене в углу, его оттуда уже не выковыряешь - раз, и два ящика толкать - силенок не хватает - два) игра, особенно на дальнейших лабиринтах, становится нетривиальной логической задачей.

Нюансами реализации, требующими особой проработки являются следующиме несколько мест в игре.

Во-первых, это различная реакция графических изображений частей игры (далее - спрайтов) на свое местонахождение. А именно, если герой находится на пустом месте, он выглядит следующим образом: ><. Если он наступает на место для ящика, он опять-таки выглядим вышеприведенным образом, в отличии от ящика, который на пустом месте выглядит <>, а на месте для ящика - шш (только повернутым на 45 градусов).

Второе (учитывающее первое) это момент инициализации различных лабиринтов в игре. Ибо в разных лабиринтах бывает разнообразие следующих ситуаций: ящик на пустом месте, ящик на месте для ящика, герой на пустом месте, герой на месте для ящика(! - попробуй внешне различить!).

Третье, проистекающее из первого и второго - это момент подсчета очков: как и каким образом его осуществлять, если учитывать, что мы пишем не только универсальный алгоритм обработки множества различных лабиринтов, но и эффективный по времени выполнения алгоритм.

Прежде всего, о программе в целом и об игрушке как программе.

В любой компьютерной игрушке (и наша не является исключением) можем мы различить следующие части:

```
      - логический драйвер устройств ввода;

      - драйвер устройств вывода;

      - структуры данных (как внутри игрушки, так и вне ее - форматы
        данных, хранящихся в файлах и т.д.);

      - основной цикл игры.
```

Логический драйвер ввода

То, что так красиво называется, является, грубо говоря, подпрограммой, а точнее выражаясь, функцией, занимающейся вводом команд с клавиатуры. Хотя в более сложных игрушках драйвер ввода обрабатывает разные устройства ввода: мыши, джойстики, птицы...

Так вот, программы уровня непосредственно обработки устройств ввода называть принято драйверами. А программы, с которыми взаимодействует игрушка, принято называть логическим драйвером ввода (понимание этих слов понадобится нам при чтении листингов).

В частности, логический драйвер унифицирует ввод, что позволяет запараллеливать при необходимости работу клавиатуры и мыши, например.

Драйвер устройств вывода

Драйвер вывода мы собираемся писать таким образом, чтобы он был универсален. А под универсальностью мы будем понимать возможность для себя с минимальными затратами вводить на любом этапе работы изменения в программу.

Структуры данных и алгоритмы их обработки

Безусловно, уже глядя на лабиринт игрушки, мы понимает, что основной структурой данных в программе будет массив, где будет закодирована информация о текущем состоянии в игре. Более тонким является вопрос, каким образом мы будем хранить информацию в этом массиве. Но не будем томить читателя последовательностью рассуждений о возникающих на каждом шагу трудностях и путях их преодоления. Перейдем к конечному результату.

Массив игры представляет собой эквивалент изображения на экране, но в кодах, что позволяет функции движения осуществлять проверку корректности перемещения по лабиринту.

Каждая позиция массива игры хранит код спрайта, который находится на соответствующем ему месте на экране. Но! Старший бит каждой позиции является признаком того, есть ли данная позиция местом для хранения ящика. таким образом, если данная координата содержит код спрайта границы лабиринта, то код ее будет... один. А если данная координата является местом для установки ящика, то код ее будет другой, но старший бит данной позиции в массиве игры будет установлен в единицу.

Далее, вводится два набора спрайтов. Во втором код каждого спрайта имеет установлен старший бит. При такой кодировке функция вывода спрайта на экран пишется для каждого набора спрайтов.

Что позволяет успешно преодолеть первую из перечисленных нами трудностей. Функция вывода спрайта будет выводить для кода героя и кода героя с установленным старшим битом одно и то же изображение, а для кода ящика и кода ящика, установленного на место - разные изображения.

Похожая кодировка позволяет преодолеть и вторую нашу трудность, связанную с инициализацией следующего лабиринта. Один код мы используем для кодирования героя, стоящего на пустом месте, другой - для героя, который стоит на месте, предназначенном для установки ящика.

Причем, при инициализации лабиринта мы используем два счетчика в игрушке: общее количество мест для установки ящиков и количество уже установленных на место ящиков. Если при этом учеть еще и принцип, по которому мы собирается обрабатывать визуализацию перемещения - если по координате, на которой только что стоял ящик, пишется не-ящик, уменьшить счетчик количества поставленных на место ящиков; если же по координате, на которой должен стоять ящик, пишется код ящика, то увеличить счетчик количество послравленных на место ящиков - то становится ясным, как божий день, как определить завершение игры: сравнить значение обоих счетчиков. При равенстве значений, все ящики установлены на предназначенные для них места. Что, собственно говоря, и позволяет преодолеть нам третью трудность, встреченную на пути реализации игрушки.

И, перед тем, как перейти к собственно программам, еще одно слово об обработке движений главного героя.

Процедура перемещения главного героя осуществляет обработку движения в заданном направлении. Направление задается парой смещений: дельта-X и дельта-Y, которые позволяют одному алгоритму обрабатывать сразу четыре направления (как выразился бы поэт: одним выстрелом убить сразу четыре зайца). Так, например, если дельта-X = 0, а дельта-Y = + 1, то это означает, что отрабатывается движение в направлении вниз, то есть, согласно следующей системе координат:

```
                        ^  ( 0, -1 )
                        }
                        |
                <_______+______>
       ( -1, 0 )        }  ( + 1, 0 )
                        }
                        V  ( 0, + 1 )
```

Вот в данном направлении, оператор

```
массив_игры[ координата-X + дельта-X, координата-Y + дельта-Y ]
```

обеспечит обращение к позиции по координате в нужном направлении, а оператор

массив_игры[координата-X + дельта-X *2, координата-Y + дельта-Y* 2]

к позиции, второй от текущей по данной координате.

Вот, собственно по-существу и все. Перейдем к делу.

Паскаль

```
$MACRO RP TRANS;
{******************************MULTI-EDIT MACRO*******************

Name:  RP    ( russian pusher - altrnative name of Soko-Ban )

Description: программа продолжает славную серию реализаций игрушки
             Soko-Ban на различных языках программирования
             от Insight corp.,
             представляя Multi-Edit...

               (C) Copyright 1991 by Insight corp.
*******************************************************************


  { Несколько слов о соглашениях именования переменных в языке,
  облегчающих программирования.

  Поскольку регистр в именах переменных - неважен, имена принято на
  дополнительным смыслом. а именно:
    если все буквы в имени переменной - большие, то данная переменн
    используется как константа;
    собственно переменные в MultiEdit обозначаются одним или нескол
    словами, написанными через символ подчеркивание, все слова начи
    с большой буквы;
    если в переменной до символа подчеркивания две больших буквы, а
    - маленькие, то данная переменная будет формальным параметром в
    функции

  Def_Int( F_KEY, F_ESC, F_RESTART, F_LEFT, F_RIGHT, F_UP, F_DOWN,
       C_CTRL_BIT, C_BIT_MASK,
       S_FREE_PLACE, SB_FREE_PLACE, S_MAN, SB_MAN, S_WALL, SB_WALL,
       S_BOX, SB_BOX );

  F_KEY   := 256;

  F_ESC   := 27;              F_RESTART := 32;
  F_LEFT  := F_KEY + $4b;
  F_RIGHT := F_KEY + $4d;
  F_UP    := F_KEY + $48;
  F_DOWN  := F_KEY + $50;
```

```
  C_CTRL_BIT := $80;
  C_BIT_MASK := $7f;


  S_FREE_PLACE := 0;        SB_FREE_PLACE := C_CTRL_BIT + S_FREE_PL
  S_MAN        := 1;        SB_MAN        := C_CTRL_BIT + S_MAN;
  S_WALL       := 9;        SB_WALL       := C_CTRL_BIT + S_WALL;
  S_BOX        := 4;        SB_BOX        := C_CTRL_BIT + S_BOX;


  Def_Int( i, j,
           Man_X, Man_Y,
           Kbrd_Key,
           Score_New, Score_All
           ) ;
  Def_Int( SS_x, SS_y, SS_d, SS_type,
           MV_x, MV_y
           );

  { задаем длину массива: 20 * 31 = 620 }
  Def_Str( Maze[ 620 ], GF[ 620 ],
         SS_char[ 2 ] );


  Put_Box( 5, 4, 70, 20, RED, WHITE,
           ь Soko-Ban game /Multi-Edit version/ ь, TRUE );
  Beep;

 {-------------------------------------------------------------------

 { исходное поле игры }
 {                     0        1         2         3   }
 {                     1234567890123456789012345678901  }
       Maze :=        ь0000000000000000000000000000000ь;
       Maze := Maze + ь0000000999990000000000000000000ь;
       Maze := Maze + ь0000000900090000000000000000000ь;
       Maze := Maze + ь0000000940090000000000000000000ь;
       Maze := Maze + ь0000099900499000000000000000000ь;
       Maze := Maze + ь0000090040040900000000000000000ь;
```

```
        Maze := Maze + ь000999090999090000099999900000ь;
        Maze := Maze + ь000900090999099999990033900000ь;
        Maze := Maze + ь000904004000000000000033900000ь;
        Maze := Maze + ь000999990999909199990033900000ь;
        Maze := Maze + ь000000090000009990099999900000ь;
        Maze := Maze + ь000000099999999000000000000000ь;
        Maze := Maze + ь000000000000000000000000000000ь;

   {---------------------- считывание лабиринта -------------------
    Score_All := 0;
    Score_New := 0;

    Def_Char( Crnt );
    i := 0;
    while ( i <= 12 ) do
      j := 1;
      while ( j <= 30 ) do
        SS_x := j;
        SS_y := i;
        Crnt := Copy( Maze, j+(i*30), 1 );
             if      ( Crnt = ь0ь ) then
                     SS_d := S_FREE_PLACE;
             else if ( Crnt = ь9ь ) then
                     SS_d := S_WALL;
             else if ( Crnt = ь4ь ) then
                     SS_d := S_BOX;
             else if ( Crnt = ь3ь ) then
                     SS_d := Sb_free_place;
                     Score_All := Score_All + 1;
             else if ( Crnt = ь7ь ) then
                     SS_d := SB_BOX;
                     Score_All := Score_All + 1;
                     Score_New := Score_New + 1;
             else if ( Crnt = ь1ь ) then
                     SS_d := S_MAN;
                     Man_X := j;
                     Man_Y := i;
             else if ( Crnt = ь2ь ) then
                     SS_d := Sb_MAN;
                     Man_X := j;
```

```
                    Man_Y := i;
                    Score_All := Score_All + 1;
            end; end; end; end; end; end; end;
       call Sprite_Show;
       j := j+1;
     end;
     i := i+1;
   end;
 {-------------------------------------------------------------------

   Write( ь Left/Right/Down/Up - moving  Esc/Space - exit ь,
             14, 19, RED, WHITE );
   Beep; Beep;
   { основной цикл игрушки }

   Def_Int( Game_Yes );

   Game_Yes := TRUE;
   while ( Game_Yes ) do
     SS_x := Man_X; SS_Y := Man_Y; SS_d := S_MAN;
     call Sprite_show;

     Read_Key;   { введем символ }
     if ( Key1 = 0 ) then
          Kbrd_Key := F_KEY + key2;
     else Kbrd_Key := Key1;    end;

     SS_x := Man_X; SS_y := Man_Y; SS_d := S_FREE_PLACE;
     call Sprite_Show;

         if      ( Kbrd_Key = F_LEFT ) then
                 MV_x := -1; MV_y := 0;
                 call Moving;
         else if ( Kbrd_Key = F_RIGHT ) then
                 MV_x := 1; MV_y :=  0;
                 call Moving;
         else if ( Kbrd_Key = F_UP ) then
                 MV_x := 0; MV_y := -1;
                 call Moving;
         else if ( Kbrd_Key = F_DOWN ) then
```

```
                MV_x :=  0; MV_y := 1;
                call Moving;
        end; end; end; end;

    if ( ( Score_New = Score_All )
       or ( Kbrd_Key = F_ESC ) or ( Kbrd_Key = F_RESTART ) ) then
       Game_Yes := FALSE;
    end;
  end;
{------------------------------------------------------------------


  if ( Score_New = Score_All ) then
      Write( ьмммммммммммммм ***   маладец!  *** мммммммммммммь,
             14, 19, RED, WHITE );
  else
      Write( ьмммммммммммммм *** слаб-б-бак! *** мммммммммммммь,
             14, 19, RED, WHITE );
  end;

  Beep; Beep; Beep;
  Read_Key;

  Kill_Box;


goto End_Of_Macro;
{------------------------------------------------------------------


{************************MULTI-EDIT MACRO SUBROUTINE**************

Subroutine Name:

Subroutine Description:


******************************************************************


Sprite_Show: { подпрограмма вывода спрайта /локальная/
```

```
                SS_x - координата по горизонтали
                SS_y - координата по вертикали
                SS_d - данное }
        if ( Ascii(Copy( GF, SS_y*30+SS_x, 1) ) = SB_BOX ) then
           if ( SS_d <> S_BOX ) then
              Score_New := Score_New - 1;
        end; end;
        if ( (Ascii(Copy( GF, SS_y*30+SS_x, 1) )
           and  C_CTRL_BIT ) <> 0  )  then
           if (SS_d = S_BOX ) then
              Score_New := Score_New + 1;
        end; end;

        GF := Str_Ins( Char( (Ascii(Copy( GF, SS_y*30+SS_x, 1 ))
                                and C_CTRL_BIT ) or (SS_d) ),
                         Str_del( GF, SS_y*30+SS_x, 1),
                         SS_y*30+SS_x
                       );

        { выводим спрайт на экран /бывшая sprite_out/ }
        SS_type := Ascii( Copy( GF, SS_y*30+SS_x, 1 ) );
        if       ( SS_type = S_FREE_PLACE ) then
                SS_char := ь  ь;
        else if ( SS_type = SB_FREE_PLACE ) then
                SS_char := ь. ь;
        else if (( SS_type = S_MAN ) or (SS_type = SB_MAN )) then
                SS_char := ь><ь;
        else if (( SS_type = S_WALL ) or (SS_type = SB_WALL )) then
                SS_char := Char( 178 ) + Char( 178 );
        else if ( SS_type = S_BOX ) then
                SS_char := ь<>ь;
        else if ( SS_type = SB_BOX ) then
                SS_char := Char( 17 ) + Char( 16 );
        end; end; end; end; end; end;

        Write( SS_char, 5 +(SS_x*2), 5+SS_y, RED, WHITE );

        ret;

  {***********************MULTI-EDIT MACRO SUBROUTINE**************
```

```
Subroutine Name:

Subroutine Descroption:



*******************************************************************

Moving: { процедура верификации перемещения двигателя
             /MV_x смещение координаты по горизонтали
             /MV_y смещение координаты по вертикали
          }
      { свободно ли следующее поле }
      if ( Ascii( Copy( GF, (Man_Y+MV_y)*30+(Man_X+MV_x), 1))
         and C_BIT_MASK = S_FREE_PLACE  ) then

         Man_X := Man_X + MV_x;
         Man_Y := Man_Y + MV_y;
      else { проверим, может это ящик }
         if ( Ascii( Copy( GF, (Man_Y+MV_y)*30+(man_x+MV_x), 1))
            and C_BIT_MASK = S_BOX   ) then

            if ( Ascii( Copy( GF, (Man_Y+(MV_y*2))*30+(Man_X+(MV_x
               and C_BIT_MASK = S_FREE_PLACE ) then

               { переместить двигатель и ящик в данном направлении
               SS_x := Man_X + MV_x;
               SS_y := Man_Y + MV_y;
               SS_d := S_FREE_PLACE;
               call Sprite_Show;
               SS_x := Man_X + (MV_x*2);
               SS_y := Man_Y + (MV_y*2);
               SS_d := S_BOX;
               call Sprite_Show;
               Man_X := Man_X + MV_x;
               Man_Y := Man_Y + mv_y;
            end;
         end;
      end;
```

```
      ret;
{------------------------------------------------------------------

End_Of_Macro:
END_MACRO;
```

```
procedure rp
set talk off

*================================================ 08/18/91 05:19pm
*
*                реализация игрушки Soko-ban преследует учебные
*                   цели и акцентирует внимание на критичных
*                     к процессу программирования местах
*
*
*                ------ версия "З н а н и е  -  с и л а" ------
*
*=========================================== (c) Insight corp., 1991

*============================================== определение констант

*------------------------------------------- коды управляющих клавиш
F_esc = 27
F_restart = 32  &&  фактически - код пробела, но для нас он игра-
                        *ет роль клавиши, сбрасывающей лабиринт в н
                        *ное состояние
F_Left  = 19
F_Right =  4
F_Up    =  5
F_Down  = 24

*----------------------------константы, задающие кодировку спрайтов
   * обратим внимание на тот факт, что спрайты с индексом b      *)
   * представляют собой код, смещенный на C_ctrl_bit, что        *)
   * позволяет легко определять поле, в котором находится ьточкаь*)
   * - место, предназначенное для заталкивания туда ящиков       *)

 * свободное для движения пространство
 *  кстати, места под точками с точки зрения данного выражения
 *являются свободными для движения)

store 2 to C_free_place, S_free_place
store - S_free_place to Cb_free_place, Sb_free_place
 * фигурка двигателя (объекта, который двигает ящики)
```

```
store 1 to C_man, S_man
store - S_man to Cb_man, Sb_man
 * стена лабиринта
store 9 to C_wall, S_wall
store - S_wall to Cb_wall, Sb_wall
 * код объекта передвижения - ящика
store 4 to C_box, S_box
store - S_box to Cb_box, Sb_box

 *===================================================================

dimension    game_field ( 20, 30 )
                      * массив пред-   *)
                      * ставляет собой эквивалент изображения на
                      * экране, но в кодах, что позволяет функции
                      * moving осуществлять проверку корректности
                      * перемещения по лабиринту

man_X=0       && пара целых представляет собой текущие коор-   *)
man_Y=0       && динаты двигателя                              *)

score_all=0   && общее количество мест для ящиков              *)
score_new=0   && текущее количество ящиков, уложенных на место *)

kbrd_key=0    && переменная - код следующего введенного        *)
              && с клавиатуры символа                          *)


bell = [? chr( 7 )]   && супер-пупер-подпрограмма для выдачи сигнал

*-------------------------------------------------------------------
clear

&bell
*@ 5, 10 double
do maze_init

&bell
&bell
do game_round
```

```
*------------------------------------------------- раздача слоников
&bell
&bell
&bell
    if( score_new = score_all )
        @ 16, 20 say [*** маладец! ***]
    else
        @ 16, 20 say [*** слаб-ба-к! ***]
    endif
*-------------------------------------------------------------------
set console off
wait
set console on

clear
return      && -- end of procedure rp --




*=====================функция выводит на экран изображение спрайта

function sprtout
parameters sprite_code
        *-----------------------------------------------------------
        * благодаря тому, что вся программа пользуется для вывода
        * спрайта на экран этой функцией, здесь
        * локализуется информация о внешнем виде спрайта,
        * его цвете, что соответственно, облегчает задачу
        * проведения изменений в игрушке
        *-----------------------------------------------------------
do case
   case (sprite_code = C_free_place)
        sprite_pict = [  ]
   case sprite_code = Cb_free_place
        sprite_pict = [. ]
   case (sprite_code = C_man) .or. (sprite_code = Cb_man)
        sprite_pict = [><]
   case (sprite_code = C_wall) .or. (sprite_code = Cb_wall)
        sprite_pict = chr(178)+chr(178)
```

```
     *------------------------------------------------------------------
     *| интерес представляет тот факт, что по кодам S_man и  Sb_man
     *| выводится один и тот же рисунок спрайта, а по кодам S_box и
     *| Sb_box - разные, что дает возможность автоматически выводит
     *| на экран правильную картинку для ящика (незакрашенный, если
     *| на пустом поле и закрашенный, если на поле, предназначенном
     *| ящика) и двигателя ( одну и ту же картинку и на пустом поле
     *| и на поле для ящика
     *------------------------------------------------------------------
   case (sprite_code = C_box)
        sprite_pict = [<>]
   case (sprite_code = Cb_box)
        sprite_pict = chr(17)+chr(16)
   otherwise
        sprite_pict = [  ]
endcase
return( sprite_pict )

*===================================================================

procedure sprite_show
parameters sprite_x, sprite_y, sprite_type
        *-----------------------------------------------------------
        * процедура обеспечивает логический вывод графического обра
        * за спрайта в пределах всех программы, здесь же происходит
        * коррекция содержимого массива game_field[];
        *
        * в данной процедуре поддерживается ведение счетчика игры
        *-----------------------------------------------------------

     * проверка необходимости корректировать счетчик очков *)
     if( game_field( sprite_x, sprite_y ) = Cb_box )
          if( sprite_type <> C_box )
            * если по координате, на которой только что стоял ящик,
            *пишется не-ящик, уменьшить счетчик количества поставле
            *на место ящиков
                         score_new = score_new - 1
           endif
     endif
```

```
     if( game_field( sprite_x, sprite_y ) < 0 )
          if( sprite_type = C_box )
            * если по координате, на которой должен стоять ящик,
            *пишется код ящика, увеличить счетчик количества постав
            *ленных на место ящиков
                 score_new = score_new + 1
          endif
     endif

     game_field( sprite_x, sprite_y ) = ;
         sign( game_field( sprite_x, sprite_y )) * ;
             abs( sprite_type )
   if( sprite_type < 0 )
       game_field( sprite_x, sprite_y ) = ;
          - game_field( sprite_x, sprite_y )
    endif

     @ sprite_x, sprite_y*2 ;
         say sprtout( game_field( sprite_x, sprite_y ) )
return


*=============================================================================

procedure maze_init
        *-----------------------------------------------------------
        * процедура инициализирует массив game_field[], содержащий
        * поле лабиринта, ящики, места для них
        * а также устанавливает счетчики игрушки: общее количество
        * мест для установки ящиков score_all, количество уже
        * установленных на место ящиков score_new
        *-----------------------------------------------------------

dimension maze( 20 )

*                     0         1         2         3
*                     1234567890123456789012345678901
        maze( 01 ) = ь00000000000000000000000000000ь
        maze( 02 ) = ь00000009999000000000000000000ь
        maze( 03 ) = ь00000009000900000000000000000ь
```

```
        maze( 04 ) = ь000000094009000000000000000000ь
          maze( 05 ) = ь000009990049990000000000000000ь
        maze( 06 ) = ь000009004004090000000000000000ь
        maze( 07 ) = ь000999090999090000099999900000ь
        maze( 08 ) = ь000900090999099999990033900000ь
        maze( 09 ) = ь000904004000000000000033900000ь
        maze( 10 ) = ь000999990999909199990033900000ь
        maze( 11 ) = ь000000090000009990099999900000ь
        maze( 12 ) = ь000000099999999000000000000000ь
        maze( 13 ) = ь000000000000000000000000000000ь

*---------------- начальная инициализация массива & очистка экрана

    i = 1
    do while( i <= 20 )
       j = 1
       do while( j <= 30 )
          game_field( i,j ) = C_free_place
          j = j + 1
        enddo
        i = i + 1
     enddo

*-------------------------------------------- считывание лабиринта

    score_all = 0
    score_new = 0

    i = 1
    do while( i <= 13 )
       j = 1
       do while( j <= 30 )
          do case
             case substr( maze( i ), j, 1 ) = ь0ь
                  do sprite_show with i, j, S_free_place
             case substr( maze( i ), j, 1 ) = ь9ь
                  do sprite_show with i, j, S_wall
             case substr( maze( i ), j, 1 ) = ь4ь
                  do sprite_show with i, j, S_box
             case substr( maze( i ), j, 1 ) = ь3ь
```

```
                    do sprite_show with i, j, Sb_free_place
                    score_all = score_all + 1
               case substr( maze( i ), j, 1 ) = ь7ь
                    do sprite_show with i, j, Sb_box
                    score_all = score_all + 1
                    score_new = score_new + 1
               case substr( maze( i ), j, 1 ) = ь1ь
                    do sprite_show with i, j, S_man
                    man_X = i
                    man_Y = j
               case substr( maze( i ), j, 1 ) = ь2ь
                    do sprite_show with i, j, Sb_man
                    man_X = i
                    man_Y = j
                    score_all = score_all + 1
               otherwise
                    do sprite_show with i, j, S_free_place
              endcase
            j = j + 1
          enddo
          i = i + 1
       enddo

return

*==================================================================

procedure moving
parameters dlt_x, dlt_Y
        *------------------------------------------------------------
        * процедура осуществляет отработку движения в заданном напр
        * лении
        *
        * направление задается парой смещений dlt_x, dlt_y
        *
        * так, например, если dlt_x = 0, а dlt_y = + 1, то это озна
        * ет, что отрабатывается движение           ^
        * в направлении вниз, то есть,              | ( 0, - 1 )
        * согласно следующей системе                |
        * координат:                        <-- -- + -- -->
```

```
        *                              ( - 1, 0 )  |     ( + 1, 0
        *                                          |
        *                                          V ( 0, + 1 )
        *-----------------------------------------------------------

        * свободно ли следующее поле?

if( abs(game_field(man_X+dlt_X,man_Y+dlt_Y)) = S_free_place)
          * перемещение двигателя на следующее поле
          man_X = man_X + dlt_X
          man_Y = man_Y + dlt_Y
else && проверим, может это ящик
    if( abs(game_field(man_X+dlt_X, man_Y+dlt_Y)) = S_box )
        * а свободно ли поле за ящиком?
        if( abs(game_field(man_X+dlt_X*2, man_Y+dlt_Y*2)) ;
                           = S_free_place )
             * переместим ящик и двигателя на поле в данном направл
             do sprite_show with man_X+dlt_X, man_Y+dlt_Y, S_free_p
             do sprite_show with man_X+dlt_X*2, man_Y+dlt_Y*2, S_bo
             man_X = man_X + dlt_X
             man_Y = man_Y + dlt_Y
         endif
    endif
endif
return

*===================================================================

procedure game_round

        *-----------------------------------------------------------
        *
        *                основной цикл игрушки
        *
        *-----------------------------------------------------------

game_yes = .t.

do while( game_yes )
```

```
    * выводим двигателя по текущей координате
do sprite_show with man_X, man_Y, S_man

        * вводим команду с клавиатуры
* сразу же заметим, что пользоваться командой inkey() в данном
* случае (то есть в данном языке) надо осторожно: здесь эта ком
* не вводит символ с клавиатуры, а производит опрос клавиатуры
* последующим вводом символа
kbrd_key = 0
DO WHILE .NOT.( (kbrd_key = F_esc).or.(kbrd_key = F_restart).or
                (kbrd_key = F_Left).or.(kbrd_key = F_Right).or.
                (kbrd_key = F_Up).or.(kbrd_key = F_Down) )
   kbrd_key = 0
   DO WHILE kbrd_key = 0
      kbrd_key = INKEY()
   ENDDO
ENDDO




    * стираем двигателя по текущей координате
do sprite_show with man_X, man_Y, S_free_place

    * отрабатываем управляющую клавишу
do case
   case (kbrd_key = F_Left )
        do moving with 0, -1

   case (kbrd_key = F_Right )
        do moving with 0, +1

   case (kbrd_key = F_Up )
        do moving with -1, 0

   case (kbrd_key = F_Down )
        do moving with +1, 0

endcase

* и так до тех пор, пока либо не установим все ящики на месте,
```

```
    *не введем код F_esc, либо код F_restart

  if(  (score_new = score_all) ;
         .or. (kbrd_key = F_esc) .or. (kbrd_key = F_restart) )
      game_yes = .f.
   endif
enddo

return
```

```
/////////////////////////  rp.slt - russian pusher ///////////////

//
//   - Written.....: by Insight corp.
//   - Date........:    09/04/91 02:24am
//   - Subject.....: программа продолжает славную серию реализаций
//       Soko-Ban на различных языках программирования,
//       представляя язык SALT телекоммуникационного пакета Telix..
//   - Compile line: cs rp

//   ---------------------------------------------------------------

///////////////////////////////////////////////////////////////////

int f_esc       = 0x001b,
    f_restart   = 0x0020;

int f_left      = 0x4b00,
    f_right     = 0x4d00,
    f_up        = 0x4800,
    f_down      = 0x5000;

//               15 + (16 * 04)  -- белым по красному
int game_color  = 0x4f;

int c_cntrl_bit = 0x80,
    c_bit_mask  = 0x7f;

int s_free_place,  s_man,  s_wall,  s_box,
    sb_free_place, sb_man, sb_wall, sb_box;


//  30 * 13 = 390
str maze[ 390 ];    // фактически - локальная переменная, но ограни
                    // длины локальных переменных до 255 символов
                    // вынуждает вынести ее на глобальное место

str game_field[ 390 ]; // массив содержит рабочий массив игрушки
```

```
int man_x, man_y,   // координаты двигателя

   score_all, score_new, // переменные содержат счетчики очков

   kbrd_key;        // введенный символ

//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\
main()
{
 int work_frame;

    // фактически - инициализация констант, но невозможность
    // использовать при инициализации выражения вынуждает
    // инициировать эти константы здесь
    s_free_place = 0;      sb_free_place = c_cntrl_bit + s_free_pla
    s_man = 1;             sb_man = c_cntrl_bit + s_man;
    s_wall = 9;            sb_wall = c_cntrl_bit + s_wall;
    s_box = 4;             sb_box = c_cntrl_bit + s_box;

 cursor_onoff ( 0 );
 work_frame = vsavearea ( 5, 4, 70, 20 );
 box ( 5, 4, 70, 20, 2, 0, game_color );

 alarm ( 1 );



 pstraxy ( "(c) Insight corp., 1991", 25, 16, game_color );
 pstraxy ( "Init iNit inIt iniT....", 25, 18, game_color );
 maze_init ();

 pstraxy ( "Moving Up/Down/Left/Right : Finish Esc/Space",
           18, 18, game_color );
 alarm ( 1 );
 game_round ();

 pstraxy ( "                                            " ,
           18, 18, game_color );
 gotoxy ( 28, 18 );
 if ( score_new == score_all )
```

```
    pstra ( "*** маладец! ***", game_color );
  else
    pstra ( "*** слаб-ба-к ***", game_color );
  alarm ( 1 );

  kbrd_key = inkeyw ();
  vrstrarea ( work_frame );
  cursor_onoff ( 1 );
 }
//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\//\



//------------------//---------------------//------------------/

// -------- функции сконструированные для работы с массивами -----
// выбрать элемент из массива array по координате [colunm, row]
item_from_array ( str array, int column, int row )
{
  return ( subchr ( array, row*30+column ) );
}

// поместить данное item в массив array по координате [column, row]
item_to_array ( str array, int column, int row, int item )
{
   setchr ( array, row*30+column, item );
}

//------------------//---------------------//------------------/
// функция выводит на экран изображение спрайта
sprite_out ( int sprite_type )
{
  if ( sprite_type == s_free_place )
    pstra ( "  ", game_color );
  else if ( sprite_type == sb_free_place )
    pstra ( ". ", game_color );
  else if ( (sprite_type == s_man) or (sprite_type == sb_man) )
    pstra ( "><", game_color );
  else if ( (sprite_type == s_wall) or (sprite_type == sb_wall) )
    pstra ( "^178^178", game_color );
```

```
  else if ( sprite_type == s_box )
    pstra ( "<>", game_color );
  else if ( sprite_type == sb_box )
    pstra ( "^017^016", game_color );
}

//------------------//---------------------//------------------//
//
sprite_show ( int sprite_x, int sprite_y, int sprite_type )
{
  if ( item_from_array ( game_field, sprite_x, sprite_y ) == sb_box
    if ( sprite_type != s_box )
      --score_new;

  if ( (item_from_array ( game_field, sprite_x, sprite_y )
                           & c_cntrl_bit ) != 0 )
    if (sprite_type == s_box )
      ++score_new;

  item_to_array ( game_field, sprite_x, sprite_y ,
    (item_from_array( game_field, sprite_x, sprite_y) & c_cntrl_bit)
    | (sprite_type) );

  // 5+3, 4+1 - смещения в home при выводе в окнышко
  gotoxy ( 5+3 + sprite_x*2, 4+1 + sprite_y );
  sprite_out ( item_from_array (game_field, sprite_x, sprite_y ));
}

//------------------//---------------------//------------------//
//
maze_init ()

{
  int i, j, crnt_char;

   //                   0        1         2         3
   //                   1234567890123456789012345678901
        maze         = "0000000000000000000000000000000" ;
        strcat ( maze, "0000000999900000000000000000000" );
        strcat ( maze, "0000000900090000000000000000000" );
```

```
       strcat ( maze, "000000094009000000000000000000" );
       strcat ( maze, "000009990049990000000000000000" );
       strcat ( maze, "000009004004090000000000000000" );
       strcat ( maze, "000999090999090000099999900000" );
       strcat ( maze, "000900090999099999990033900000" );
       strcat ( maze, "000904004000000000000033900000" );
       strcat ( maze, "000999990999909199990033900000" );
       strcat ( maze, "000000090000009990099999900000" );
       strcat ( maze, "000000099999999000000000000000" );
       strcat ( maze, "000000000000000000000000000000" );

// ---------------------------------- считывание лабиринта-------
score_all = 0;
score_new = 0;

for ( i = 0; i < 13; ++i )
  for ( j = 0; j < 30; ++j )
    {
     item_to_array ( game_field, j, i, 0 );
         // сначала избавимся от мусора...
     crnt_char = item_from_array ( maze, j, i );
         // а теперь - от необходимости повторять правое выражение
         // в каждом if-е
     if ( crnt_char == ь0ь )
       sprite_show ( j, i, s_free_place );
     else if (crnt_char == ь9ь)
       sprite_show ( j, i, s_wall );
     else if (crnt_char == ь4ь)
       sprite_show ( j, i, s_box );
     else if (crnt_char == ь3ь)
       {
        sprite_show ( j, i, sb_free_place );
        ++score_all;
       }
     else if (crnt_char == ь7ь)
       {
        sprite_show ( j, i, sb_box );
        ++score_all;
        ++score_new;
       }
```

```
      else if (crnt_char == ь1ь)
        {
         sprite_show ( j, i, s_man );
         man_x = j;
         man_y = i;
        }
      else if (crnt_char == ь2ь)
        {
         sprite_show ( j, i, sb_man );
         man_x = j;
         man_y = i;
         ++score_all;
        }
      else
        sprite_show ( j, i, s_free_place );
     }
}

//-------------------//---------------------//-------------------/
//
moving ( int dlt_x, int dlt_y )
{
       // а вот здесь обратим внимание на наличие скобок в выражени
       // (SALT - это тот самый язык, где лишняя пара скобок не
       // повредит (sic! rules of precedence (то бишь, правила
       // предшествования)))

 // свободно ли следующее поле?
 if ( (item_from_array( game_field, man_x+dlt_x, man_y+dlt_y)
      & c_bit_mask) == s_free_place )
   // перемещение двигателя на следующее поле
   {
    man_x = man_x + dlt_x;
    man_y = man_y + dlt_y;
   }
 else
   // проверим, может это - ящик?
   if ( (item_from_array ( game_field, man_x+dlt_x, man_y+dlt_y)
        & c_bit_mask) == s_box )
     // свободно ли поле за ящиком?
```

```
     if ( (item_from_array( game_field, man_x+(dlt_x*2),
          man_y+(dlt_y*2)) & c_bit_mask) == s_free_place )
       // переместим ящик и двигателя на поле в данном направлении
       {
        sprite_show ( man_x+dlt_x, man_y+dlt_y, s_free_place );
        sprite_show ( man_x+(dlt_x*2), man_y+(dlt_y*2), s_box );
        man_x = man_x + dlt_x;
        man_y = man_y + dlt_y;
       }
}

//--------------------//----------------------//--------------------/
//
game_round ()
{
 do {
   sprite_show ( man_x, man_y, s_man );
   kbrd_key = inkeyw ();

   sprite_show ( man_x, man_y, s_free_place );

   if ( kbrd_key == f_left )
     moving ( -1, 0 );
   else if (kbrd_key == f_right)
     moving ( 1, 0 );
   else if (kbrd_key == f_up)
     moving ( 0, -1 );
   else if (kbrd_key == f_down)
     moving ( 0, 1 );

  } while ( !( (score_new == score_all) or
             (kbrd_key == f_esc) or
             (kbrd_key == f_restart) ) );
}
//////////////////////////// the end ///////////////////////////////
```

```
      Редактор панорам
      для игрушки "Капитан Комик"
                                                          версия 2.02
                                            (c) Insight corp., 1991
                                                     11/12/91 00:31am

      Редактор панорам обеспечивает полноэкранное редактирование
```

панорам вышеупомянутой игрушки при помощи нижеуказанных команд.

```
      Содержание руководства

              Порядок работы. Вход в редактор

              Принцип реализации панорамного движения в игре

              Режим работы с кубиками изображения
                      Хождение по кубикам
                      Как сделать кубик активным
                      Вписывание (извините за выражение) кубика в
                        нужное место
              Разметка прозрачности кубиков

              Режим редактирование кубика
                      Работа с палитрой цветов
                      Завершение работы в режиме

              Порядок выхода из редактора



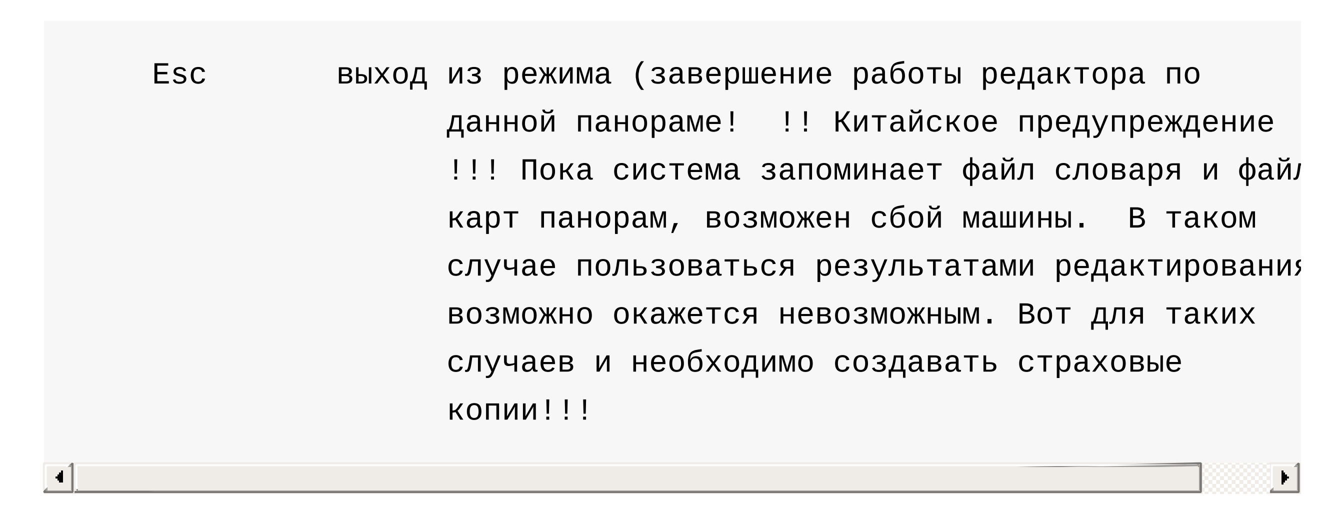
      Краткая сводка команд редактора панорам
```

Вход в редактор:

```
      > paned <Enter>
```

После загрузки редактор просит вас назвать имя панорамы, и загружает файл со словарем панорамы, а также все три карты панорам.

После чего система переходит в режим редактирования панорамы.



```
Esc       выход из режима (завершение работы редактора по
                данной панораме!  !! Китайское предупреждение
                !!! Пока система запоминает файл словаря и файл
                карт панорам, возможен сбой машины.  В таком
                случае пользоваться результатами редактирования
                возможно окажется невозможным. Вот для таких
                случаев и необходимо создавать страховые
                копии!!!
```

В данном режиме команды движения:

```
Home      панораму к левому краю
End       панораму к правому краю
CtrlPgDn  панораму вправо на кубик
PgDn      панораму вправо на экран
CtrlPgUp  панораму влево на кубик
PgUp      панораму влево на экран
Up        рамочку вверх на кубик
Down      рамочку вниз на кубик
Left      рамочку влево на кубик
Right     рамочку вправо на кубик
```

Функциональные ключи:

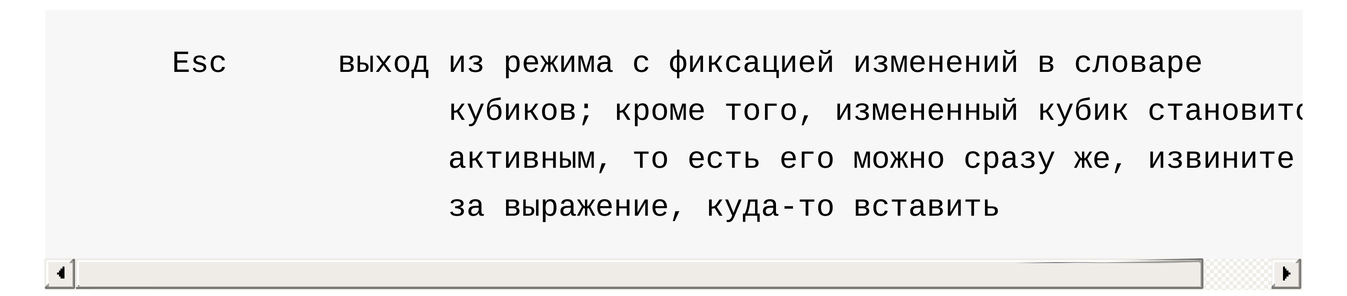
```
F2        сделать текущей карту 0
F3        сделать текущей карту 1
F4        сделать текущей карту 2
F5        фиксирование кубика как текущего по данной координате
F6        занесение текущего кубика в данную координату
F8        редактирование кубика под курсором
                (см. список команд данного режима ниже)
F7        переключить признак кубика
                прозрачный.непрозрачный
F9        переключатель режима прозрачности
CtrlF5    вызов словаря кубиков (см. список команд данного
                режима ниже)
```

В режиме индикации прозрачности (ключ F9) редактор отображает непрозрачные кубики при помощи квадратика размером ширина кубика-2 на высота кубика-2, что позволяет ориентироваться в общем плане построения препятствий. Работа системы несколько (очень незаметно, но) замедляется, а в остальном все - то же самое.

Режим редактирования изображения кубика (ключ F8):

Команды данного режима составляют три категории:

Команда выхода из режима:

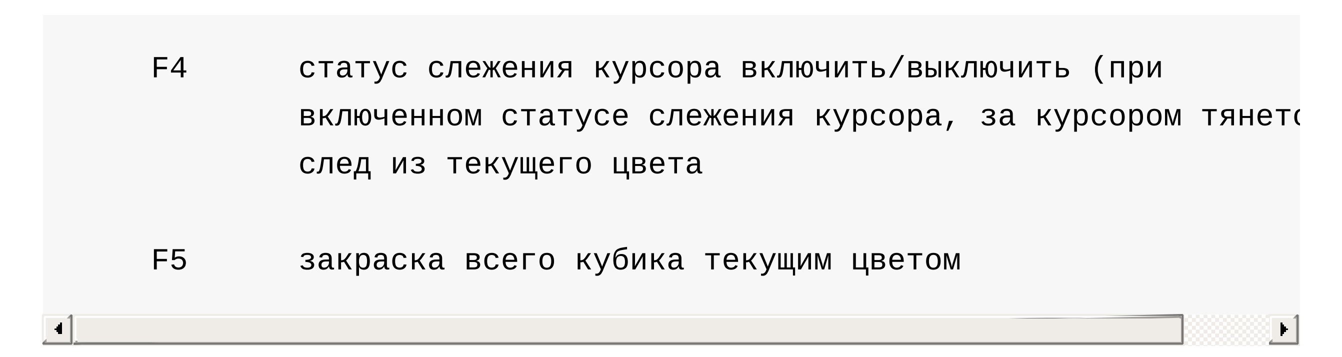


```
 Esc      выход из режима с фиксацией изменений в словаре
                кубиков; кроме того, измененный кубик становитс
                активным, то есть его можно сразу же, извините
                за выражение, куда-то вставить
```

Команды движения в режиме:

```
Left      движение влево на пиксел
Right     движение вправо на пиксел
Up        движение вверх на пиксел
Down      движение вниз на пиксел
```

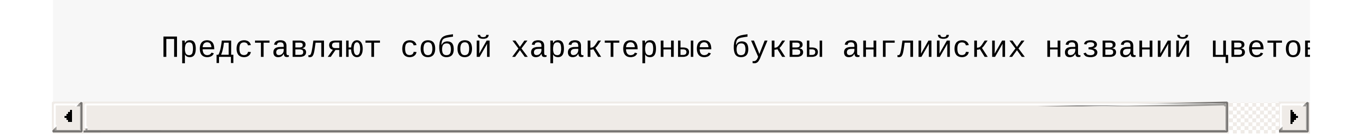
Команды движения в данном режиме отрабатывают движение по-разному в зовисимости от статуса слежения. Статус слежения представляет собой полоску желтого цвета, расположенную внизу экрана редактирования кубика (статус переключается при помощи ключа F4). Если статус слежения выключен, курсор пискеля просто двигается, если статус слежения включен, то при движении закрашивается пиксел, из которого мы только что двинулись дальше.



```
F4      статус слежения курсора включить/выключить (при
        включенном статусе слежения курсора, за курсором тянетс
        след из текущего цвета

F5      закраска всего кубика текущим цветом
```

Команды выбора цвета палитры:



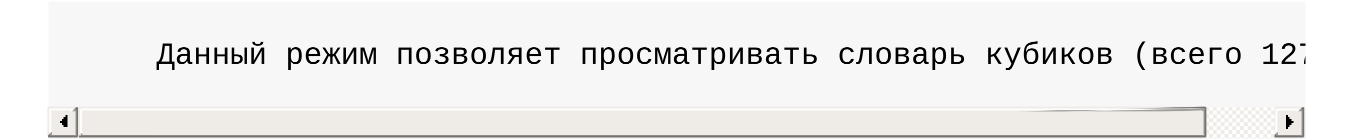
```
Представляют собой характерные буквы английских названий цветов
```

(иногда - зашифтенные (то есть, требующие нажатия шифта (то есть, клавиши, изменяющией регистр на клавиатуре)))

```
    black         черный             a
    blue          синий              b
    green         зеленый            g
    cyan          бирюзовый          c
    red           красный            r
    magenta       фиолетовый         m
    brown         коричневый         n
    white         белый              w
    gray          серый              A
    lightblue     голубой            B
    lightgreen    светло-зеленый     G
    lightcyan     светло-бирюзовый   C
    lightred      светло-красный     R
    lightmagenta  светло-фиолетовый  M
    lightyellow   желтый             Y
    brightwhite   ярко-белый         W

       Примечание: маленькие латинские буквы соответствуют нажатию
                   без шифта, большие - соотвтественно - с шифтом
```

Режим просмотра словаря кубиков (ключ CtrlF5)



```
    Данный режим позволяет просматривать словарь кубиков (всего 127
```

штук), и выбирать, либо просто нужный кубик, либо отыскивать еще не занятый картинкой (то есть пустой кубик).

После нажатия ключа, под основным экраном панорамы появляется строчка словаря кубиков (следующий ряд из 16 кубиков). При помощи команд движения мы вибираем нужный кубик и нажимаем либо клавишу Enter (и тогда после выхода из режима данный кубик становится активным) либо клавишу Esc, и тогда ничего не меняется.

Команды завершения режима:

```
    Esc       выйти из режима без ничего
    Enter     выйти из режима и слелать выбранный кубик активным
```

Команды движения в режиме:

```
    Home      к первому кубику
    End       к последнему кубику
              !! Обратите Внимание !! Последним является не 128, а
              127 кубик, что и отображается при индикации последнеı
              кубика (предпоследний в данном ряду). Если вы выбереı
              самый последний кубик, то рискуете получить потом на
              картинке мусор.
    Up        циклически на ряд вперед
    Down      циклически на ряд назад
    Left      циклически влево на кубик
    Right     циклически вправо на кубик

  зддддддддддддддддддддддддддддддддддддддддддддддддддддддддддддддд©
  Ё                                                               Ё
  Ё   Не надо забывать культурно работать со страховыми копиями   Ё
  Ё          панорам!                                             Ё
  Ё                                                               Ё
  Ё   Лучше лишний раз выйти, чем потерять несколько часов        Ё
  Ё          работы!                                              Ё
  Ё                                                               Ё
  юддддддддддддддддддддддддддддддддддддддддддддддддддддддддддддддды
```

```
                                (c) Insight corp., 1991
```

Файлы, что находятся в данном подкаталоге представляют собой пример реализации виодеоэффекта объемного движения из книги Технология Проектирования Компьютерных Игр

Для запуска видеоэффекта, выполнить файл pn.exe

Файл pn.exe использует как графические данные в процессе работы файлы pan.tt2 и pan0.pt. Чтобы поработать с этими файлами, запустите редактор paned100.exe (не 100, а версия 1.00!)

```
      Обьяснение, облегчающее разбирательство с работой редактора и
      принципами функционирования видеоэффекта:

      В первой части панорамы (при просмотре ее редактором
      paned100.exe), вверху находятся полосы, движущиеся в
      определенном порядке с разными смещениями. Внизу панорамы
      находятся полосы, перекрывающие при движении фон. Справа от
      полос, перекрывающих фон находится маска для накладки на фон.

      Остальные детали работы очевидны и докладно описаны в виде
      программы на языке C в соответствующем файле.
```